ECS417U Project Phase 2 Documentation

|  |  |
| --- | --- |
| **Name:** | Luka FB Lepkowski |
| **Student Number:** | 230409305 |

|  |  |
| --- | --- |
| **Assessment Criteria 7**  Each time a new entry is added, the most recent post appears on top, followed by the next most recent post using a PHP-driven technique (writing a sorting algorithm) instead of a SQL Query. | |
| Instruction: use this column to paste the code from the files that have been used to develop this functionality.  **Paste the code directly from visual studio code, do not include a screenshot of the code as this document has a Turnitin requirement and this will lead to an automatic mark of zero. See example below.** | Instruction: use this column to address the following in your own words:   1. Describe the steps you took to develop this feature (including the prompts). 2. Describe how the sorting algorithm works.   Give reference to the code you have written when answering the questions above.  You can use diagrams (e.g. flow charts) or screenshots of your webpage output to illustrate your answer. |
| Filename: viewBlog\_exerpt.php **(THIS IS ONLY THE PART SHOWN TO CHATGPT)**  session\_start();      $servername = "localhost";      $username = "root";      $password = "";      $dbname = "website";      #Connect      $conn = new mysqli($servername, $username, $password, $dbname);      #Check connection is okay      if($conn->connect\_error) {          exit("Connection failed: " . $conn->connect\_error);      }        #Retrieve posts      $sql = $conn->prepare("SELECT \* FROM posts");      $sql->execute();      $result = $sql->get\_result();      $rows = [];      if($result->num\_rows > 0) {          //generate array          while($row = $result->fetch\_assoc()) {              $\_sql = $conn->prepare("SELECT email FROM users WHERE user\_id = ?");              $\_sql->bind\_param('i', $row['author\_id']);              $\_sql->execute();              $\_result = $\_sql->get\_result();              if($\_result->num\_rows == 1) {                  $r = $\_result->fetch\_assoc();                  $row['author'] = $r['email'];              }              array\_push($rows, $row);          }      }      #If there's no blog, go home !      if(count($rows) == 0) {          header("Location: ");          exit();      } | Below is the transcript of the dialogue between me and ChatGPT. My prompt is highlighted in yellow.   * [viewBlog\_exerpt.php] This PHP code generates an array of $row objects called $rows. I want to have this array be sorted. Please write a sorting algorithm which sorts these rows by post date. The post date is under $row["timestamp"]. Please fully develop a sorting algorithm, rather than using a built-in sort function. Please develop a quick sorting algorithm with a low time complexity. * Sure! Here's how you can implement **Quick Sort** manually in PHP to sort your $rows array by the "timestamp" field without using built-in sorting functions like usort. * We'll define a custom quickSortByTimestamp function that recursively sorts the array using the quicksort technique. * You can add the sorting logic at the end of your code like this:   # QuickSort Algorithm by timestamp  function quickSortByTimestamp($array) {  if (count($array) < 2) {  return $array;  }  $pivot\_index = floor(count($array) / 2);  $pivot = $array[$pivot\_index]["timestamp"];  $pivot\_row = $array[$pivot\_index];    $left = [];  $right = [];    foreach ($array as $index => $item) {  if ($index == $pivot\_index) continue; // Skip the pivot  if ($item["timestamp"] <= $pivot) {  $left[] = $item;  } else {  $right[] = $item;  }  }  return array\_merge(  quickSortByTimestamp($right), // Sort DESC: newer first  [$pivot\_row],  quickSortByTimestamp($left)  );  }  # Sort the rows  $rows = quickSortByTimestamp($rows);   * This version sorts posts from **newest to oldest** (timestamp DESC). If you want it oldest first, just flip the comparison from <= to > in the if condition. * Quick sort is efficient with an average time complexity of **O(n log n)**. * Let me know if you’d like the sort order flipped or the timestamp formatting adjusted!   This ends the transcript.  **But how does it work ?**  It works like quicksort normally does. The list is split into three: A singular pivot, a left array and a right array. Everything in the left list is below the pivot (in this case by timestamp) and everything in the right list is above the pivot. Those lists are then sorted in much the same way, and then the lists are rejoined. This is a recursive sorting algorithm, as part of the process of quicksort involves calling two smaller quicksorts. This has time complexity O(n log n) as ChatGPT has kindly informed me. That’s better than n^2, which is bubble sort for example. |
|  |  |

|  |  |
| --- | --- |
| **Assessment Criteria 8 - Extra Feature 1**   * Add a “preview” button in addpost. When this button is clicked, the new entry is previewed, and you can then decide (via a set of navigational links) whether to upload the entry or go back to edit it. | |
| Instruction: use this column to paste the code from the files that have been used to develop this functionality.  **Paste the code directly from visual studio code, do not include a screenshot of the code as this document has a Turnitin requirement and this will lead to an automatic mark of zero. See example below.** | Instruction: use this column to address the following in your own words:   1. How did you develop this feature? Describe the steps involved. 2. Describe the inner workings of the approach you developed.   Give reference to the code you have written when answering the questions above.  You can use diagrams (e.g. flow charts) or screenshots of your webpage output to illustrate your answer. |
| Filename: addEntry.js  previewing = false;  function ordinal(n) {      if(n >= 11 && n <= 13) {          return n+'th';      }      switch(n % 10) {          case 1: return n+'st';          case 2: return n+'nd';          case 3: return n+'rd';          default: return n+'th';      }  }  function getNow() {      date = new Date();      day = date.getDate();        formattedDate = new Intl.DateTimeFormat('en-US', {          year: 'numeric',          month: 'long',          day: 'numeric',          hour: '2-digit',          minute: '2-digit',          hour12: false          timeZone: 'UTC'      }).format(date);      return (`${formattedDate.replace(day, `${ordinal(day)}`)} UTC`).replace(" at "," ");  }  function preview() {      title = document.getElementById("title").value.trim();      content = document.getElementById("content").value.trim();      if(!previewing) {          if(!title || !content) {              alert("Blank title/content not allowed.");              return;          }      }      previewing = !previewing;      p1 = document.getElementById("preview1");      p2 = document.getElementById("preview2");      if(previewing) {          p1.classList.remove("hidden");          p2.classList.add("hidden");          document.getElementsByClassName("blogpost-title")[0].firstChild.nodeValue = title;          document.getElementsByClassName("blogpost-date")[0].firstChild.nodeValue = "Written "+getNow();          document.getElementsByClassName("blogpost-content")[0].firstChild.nodeValue = content;      } else {          p1.classList.add("hidden");          p2.classList.remove("hidden");      }  }  …  document.addEventListener("DOMContentLoaded", function(){      previewButton = document.getElementById("preview");      previewButton.addEventListener("click", preview);  …  }  Filename: addEntry.php  …  <div class="hidden" id="preview1">                  <article class="blogpost">                      <div class="blogpost-header">                          <div class="blogpost-title">                              Title                          </div>                          <div class="blogpost-date">                              Written                          </div>                      </div>                        <div class="blogpost-content">                          Content                      </div>                  </article>              </div>              <div id="preview2">                  <form id="form" action="addPost.php" method="POST">                      <div class="input-group">                          <label for="title">Title</label><br/>                          <input type="text" id="title" name="title">                      </div>                      <br/>                      <div class="input-group">                          <label for="content">Content</label><br/>                          <textarea type="text" id="content" name="content"></textarea>                      </div>              </div>                      <br>                      <button type="submit">Post</button>                      <button type="button" id="clear">Clear</button>                      <button type="button" id="preview">Preview</button>                  </form>          </section>  …  Filename: style.css  …    /\*      Preview  \*/  .hidden {      display: none;  } | *The code on the left is an excerpt of the file, not the entire file itself. Sections have been removed for brevity as they are not relevant to the feature being described. These removed sections can be seen in the actual code and have been replaced on the left with an ellipsis. (…)*  In addEntry.js, we have a “previewing” Boolean, starting at false. Every time the preview function is called, (after checking that there is content to preview) we negate previewing (true becomes false, false becomes true). The first time the function is called, we go to previewing, and the second time we stop previewing. (and so on and so forth.)  After determining the Boolean, we apply the “hidden” class to either “preview1” or “preview2”, and remove it from the other one. These can be seen in addEntry.php, and the hidden CSS can be seen in style.css. All this style does is stop that section from being displayed.  This means they will alternate being displayed and not being displayed, every time the function is called.  These sections, “preview1” and “preview2” are the preview box and the title-content form respectively. What this has the effect of is looking like the title-content form being replaced by a preview button. That button can be pressed again to swap back to the normal field / content section.  In order to actually fill in the contents of the preview, the current date and time has to be calculated. This is calculated using ordinal() and getNow(). Ordinal converts a number into its ordinal form (1 becomes 1st, for example), used for the day of the month (e.g. 13th April, 2025). The date and time is generated using the Date object and Intl.DateTimeFormat (thanks W3Schools !). This is chucked into the blogpost-date field. The blogpost-content and blogpost-title fields are copied over from the values in the form.  At the bottom of addEntry.js, the preview function is bound to the preview button (This is in an event listener, waiting for the content (the button, more importantly) to be loaded into the DOM)  The hidden CSS is really easy: It’s just “display: none;”. This stops whatever it’s applied to from being shown. |

|  |  |
| --- | --- |
| **Assessment Criteria 9 - Extra Feature 2**  (Note: delete any of the two features below)   * Organise your blog entries into different months. Provide a drop-down menu for the blog viewers to view the entries of different months stored in an entry archive. * Allow blog viewers to log in and add comments to the entries. You, as the administrator of the blog should be able to delete entries or comments. * Add a “preview” button in addpost. When this button is clicked, the new entry is previewed, and you can then decide (via a set of navigational links) whether to upload the entry or go back to edit it. | |
| Instruction: use this column to paste the code from the files that have been used to develop this functionality.  **Paste the code directly from visual studio code, do not include a screenshot of the code as this document has a Turnitin requirement and this will lead to an automatic mark of zero. See example below.** | Instruction: use this column to address the following in your own words:   1. How did you develop this feature? Describe the steps involved. 2. Describe the inner workings of the approach you developed.   Did you use any Generative AI tools to achieve this? If yes, can you provide the prompts you wrote to achieve this?  Give reference to the code you have written when answering the questions above.  You can use diagrams (e.g. flow charts) or screenshots of your webpage output to illustrate your answer. |
| Filename: test.php  <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>Test</title>  </head>  <body>  <h1>My first PHP page</h1>  <?php  echo "Hello World!";  ?>  </body>  </html>  Filename: test1.php  <!DOCTYPE html>  <html lang="en">  <head>  <meta charset="UTF-8">  <meta http-equiv="X-UA-Compatible" content="IE=edge">  <meta name="viewport" content="width=device-width, initial-scale=1.0">  <title>Test</title>  </head>  <body>  <h1>My first PHP page</h1>  <?php  echo "Hello World!";  ?>  </body>  </html> |  |
| **Assessment Criteria 14 - Web technologies/framework**  For this project, you have made use of HTML, CSS, JavaScript and PHP. If you were to conduct this project again, which other web technologies/frameworks would you use to develop your portfolio site? Justify your choice. | |

|  |
| --- |
| There’s definitely a few to consider, but the two that I’m the most interested in using are SQL and React.  **MySQL:**  So, I did use MySQL a bit in this project, for my database storing users and posts. I just think it’s useful. I could use it in the future to store information about web interactions (like how often people visit certain areas of the website). More obviously, each of my projects really ought to be in a database. Having everything in one database is also useful because I can have the same article show up in multiple places on my site, and I can easily edit that article once, and have the update be shown everywhere. For example, my about me article could be a blog post too, and I could specifically load that onto my frontpage, all from the same database. MySQL (or some equivalent database management system, although I am more familiar with MySQL) would let me do all of that.  **React:**  I’m not as versed in React, so I don’t know exactly how effective it truly is, but in my research (searching online, watching tutorials, chatting with other developers who use it prolifically) it looks functional. React, Like what I talk about in the MySQL section above, allows for modular site building. It uses a component-based architecture (Which is sort of what I naturally gravitate towards anyways, being a primarily OOP developer). It’s built in JS, but it’s used in place of HTML. It’s got a Virtual DOM that then affects the real DOM. Using XML, an extension for React, you can write HTML inline within the file. E.g**. *const element = <h1>Hello, world!</h1>;***  While the line prior looks like a mistake, it does work ! That’s really cool. It’s not often said, but working in a framework that I actually find enjoyable is really important. I’ll just have a better work ethic and put more effort into the details if I enjoy the process of development, and having a nice framework definitely helps that. This is far more enticing than HTML/CSS/JS. (From what I can see, though, the CSS will still have to be developed normally. It can’t do everything after all. Overall It looks like with the website I’ve got right now, rebuilding with React could definitely help clean everything up, and build a better website as I go into the future. The site I have now is not future proof in the slightest. |
|  |